Course: CSCI 331

Project: Zip Code Record Processor: Part 2

Date: 10/09/25

Team 5

---

1. Project Overview

The purpose of this project is two-fold. The first objective is to convert given csv files into length indicated, comma separated files and to use old and new buffers to read the given and new csv files. The second objective is to create and read in RAM a primary key index that can display all zip code data from through and with the command line.

1.) Sequentially process the given csv files using the buffer class from project 1.
2.) Convert the csv files to be length indicated. A prefix format will be used. The csv files will remain comma separated.
3.) Unpack and read from a length indicated zip code, zip code records.
4.) Create a new buffer to class to write and read a header record.
5.) Create and read, in RAM, a primary key index for the zip code records in tandem with the command line.

Extensive documentation of the source code is also required.

---

2. Data Description

Input CSV Fields:

Field	Description	Example
Length	Length of record	31
ZipCode	5-digit zip code	90210
City	City name	Beverly Hills
State	2-character state abbreviation	CA
Latitude	Decimal latitude	34.0901
Longitude	Decimal longitude	-118.4065

---

3. File Structure

Each file has a header record which is followed by the data records.

Component	Description
4 bytes	uint32_t — Length of header record (in bytes)
N bytes	Header record
Other	Each record: 4 byte length + record data

Example (binary layout):

[35] Zip,Placename,State,County,Lat,Long

[37] 56301,Halo,MN,Stearns,23.754,72.46890

[35] 56303,NotReal,MN,Stearns,25.754, 73.864

---

4. Classes and Data Structures

Category	Items
Class	ZipCodeRecordBuffer
Data Structure	string m_fields (fixed-size array of strings)

Accessor Methods	getZipCode, getPlaceName, getState, getCounty, getLatitude, getLongitude
File Parsing Method	ReadRecord

3.2 Length Buffer Class

3.3 Containers

---

4. Program Flow / Pseudocode

---

1. Input Processing
    - Read CSV (or converted XLSX → CSV).
    - Write length-indicated file with header + records.
2. Buffer Operations
    - Use `LengthBuffer` for sequential reading and sorting (replicates Project 1.0).
3. Header Handling
    - Use `HeaderBuffer` to manage file header (field metadata).
4. Index Construction
    - Sequentially read the data file and build an index mapping Zip → offset.

    - Write index to disk.
5. Command-Line Lookup
    - Parse arguments for flags like `-Z56301`.
    - Load index, locate record, and use `LengthBuffer` to retrieve and display.
6. Output
    - Display a full labeled Zip Code record or “not found” message.

---

5. Testing Plan

Test	**Description**	**Expected Result**
1	Process CSV sequentially using a buffer.	All records read in order
2	Process randomized XLSX (as CSV).	Sorted output identical to base CSV
3	Convert CSV → length-indicated	File opens and records unpack correctly

4	Read header record.	Header matches field names.
5	Create and read index.	Offsets map correctly to Zip codes.
6	Lookup known zip record.	Full labeled record displayed.
7	Lookup unknown zip record.	Error message displayed.
8	Verify record(s)	Same number of records in all formats

---

6. Tools and Environment

- Buffer Utilities: fixtext.h/cpp
- Complier : To be determined
- Documentation: Doxygen, prelim.txt

---

7. Assumptions and Constraints

- All input files are well-formed CSVs with the same field structure.
- Each record's first field (`Zip`) is unique (primary key).
- Files fit comfortably in memory for index creation.
- System developed in C++17 or later.
- Platform: Windows, macOS, or Linux.

---

8. Notes for Future Updates

- Add a robust error handling system
- Include Doxygen comments in all methods
- Test with large datasets